NUSC Technical Document 8815
21 February 1991

In Situ Operational Reshading of Arrays With Failed Elements: Algorithm Documentation Package

M. S. Sherrill
R. L. Streit
Submarine Sonar Department

Naval Underwater Systems Center
Newport, Rhode Island • New London, Connecticut

Preface

This document was assembled to accommodate continuing requests for the subject algorithm documentation in both hard copy and floppy disk form. The authors have fielded frequent requests for this package since the algorithm was first published in the IEEE Journal of Oceanic Engineering in January 1987.

Reviewed and Approved: 21 February 1991

F. J. Kingsbury

F. J. Kingsbury
Head, Submarine Sonar Department

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE

In Situ Optimal Reshading of Arrays with Failed Elements: Algorithm Documentation Package

5. FUNDING NUMBERS

6. AUTHOR(S)

M. S. Sherrill and R. L. Streit
Submarine Sonar Department

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Underwater Systems Center
New London Laboratory
New London, CT 06320

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

An algorithm documentation package containing: (1) a reprint of an article in the IEEE Journal of Oceanic Engineering, vol. OE-12, no. 1, Jan 87, "*In Situ* Optimal Reshading of Arrays with Failed Elements;" (2) a listing for the driver routine in program "Reshade;" and (3) a 3½ inch floppy disk containing the program "Reshade" which runs on any HP Series 200/300 microcomputer.

14. SUBJECT TERMS

Reshading, linear array
Array shading weights, optimal

15. NUMBER OF PAGES

20

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

TABLE OF CONTENTS

ALGORITHM DOCUMENTATION PACKAGE 1

GENERAL APPLICATION 1

TO RUN "RESHADE" 2

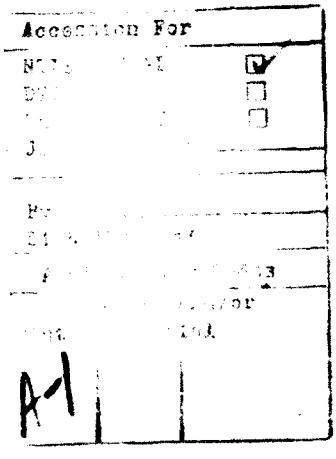
PROGRAM NOTES 2

PROGRAM EXTENSIONS AND IMPROVEMENTS 3

APPENDIX A—Reprint Paper: In Situ Optimal Reshading of Arrays with Failed Elements A-1

APPENDIX B—Driver Program Listing: "Reshade" Lines 1 through 481 ..... B-1

APPENDIX C—Floppy Disk: Program "Reshade" C-1



IN SITU OPTIMAL RESHADING OF ARRAYS WITH FAILED ELEMENTS

ALGORITHM DOCUMENTATION PACKAGE

This document assembles under one cover information on a NUSC-developed algorithm which computes optimal shading weights for discrete elements (sensors) in linear acoustic arrays. The algorithm has been found especially useful when elements fail and array reshading is required *in situ*. The main attractions of the algorithm are that it loads easily on Hewlett-Packard microcomputers, and that it runs fast enough and is accurate enough to suit most sea trial and engineering development applications. Continuing requests for this information since an invited paper first appeared in the IEEE Journal of Oceanic Engineering in January 1987 motivated the publication of this documentation package.

The information included here is in hard copy and floppy disk form: the IEEE paper, *In Situ Optimal Reshading of Arrays with Failed Elements*, is reprinted in Appendix A; a program listing of the application-specific driver routine is given in Appendix B; and a 3½ inch floppy disk, containing the program "Reshade" which runs on any Hewlett-Packard Series 200 or 300 microcomputer, is pocketed in Appendix C.

GENERAL APPLICATION

When a linear array of discrete acoustic elements is subjected to the rigors of the ocean environment, individual elements can fail. Element failures are usually characterized by noisy channels, or intermittent responses, or no response at all. Depending on the number of failed elements and their specific locations within the array, sidelobe levels of the array wavenumber (k) response can rise significantly to degrade array performance. Because element weighting values determine array wavenumber response, weights that are optimal for a fully populated array have to be recalculated when elements fail. The optimal reshading (reweighting) algorithm described here can be applied *in situ* to compute weighting values that can reduce sidelobe levels to approximately the original design specification. In fact, in the more common situations where "a few" elements fail, optimal reshading does regain original sidelobe levels. Where large numbers of elements fail, optimal reshading is still possible but may be of limited use.

The original approach for optimal reshading of a linear array was proposed by Streit and Nuttall in 1982 (see reference 1, Appendix A). At that time, the algorithm was run on a VAX 11/780 and required hours of computation time and large amounts of mass storage for rudimentary element failure problems.

The 1987 reshading algorithm incorporates several algorithmic improvements that exploit the special structure of the underlying linear programming problem to reduce time and storage requirements by orders of magnitude. The current algorithm is still based on the original theory, but is now fast enough and small enough to execute successfully in minutes instead of hours in the application environment. Execution time for a 50-element array is typically about 10 minutes. Derivation of the optimal reshading algorithm and its implementation are given in the references of the paper reprinted in Appendix A; examples of array reshading are given in the paper itself.

TO RUN "RESHADE"...

- Insert the program disk from Appendix B into device/drive.
- Type the command string **LOAD "RESHADE:*(device specifier)*"**
- Press Enter.
- When the program is loaded, press Run.
- Follow the prompts.

PROGRAM NOTES

"Reshade" comprises a driver routine in HP BASIC which sets up the necessary variables to be optimized and a generic optimization routine. The driver is listed in lines 1 through 481 of the program—the printout is contained in Appendix B.

The driver included in "Reshade" applies to a linear array of acoustic elements, some of which may have failed during the course of a sea trial or similar event. Even with the array intact, "Reshade" allows the user to minimize the sidelobe levels of the array beamformer output, given a certain mainlobe width. If the minimum sidelobe levels remain too high, it is possible to alter the mainlobe beamwidth to reduce sidelobe levels. Note that the weights on each element can be set up as non-negative, if desired.

The program prompts require user inputs, not all of which are self-explanatory. For each user input, values in (parentheses) are those allowed, and values in [brackets] are the defaults. The maximum allowable total number of array elements is 50; the minimum is three. The computation time for a 50-element array is approximately 10 minutes, while a 10-element array runs in less than one minute.

The algorithm is applicable to both equispaced or aperiodically spaced linear arrays. In the equispaced arrangement, the wavenumbers *k0* and *k1*—which delimit the region in which the minimization is performed—are calculated automatically from the desired sidelobe level. The final sidelobe level depends upon the number of failed elements in the array and their location. In this case, only the inter-element spacing must be specified.

In the aperiodically spaced arrangement, every element's position referenced to the forward end of the array must be specified. If elements have failed (or are missing), they are treated as if they do not exist. The wavenumbers *k0* and *k1* are not calculated automatically for an aperiodic array, and must be entered manually in units of radians/meter.

If unsatisfactory sidelobe levels are still present after running "Reshade" for an equispaced arrangement, *k0* can be increased to provide a larger beamwidth, thus reducing sidelobe levels. For an aperiodic array, *k0* and *k1* can be altered manually to reduce the sidelobe level in the region of interest.

Resultant weights can be stored in a data file in the following format:

- Equispaced element arrangement—total number of elements, followed by the inter-element spacing, followed by array weights.
- Aperiodic element arrangement—total number of elements, followed by each element's position, followed by array weights.

PROGRAM EXTENSIONS AND IMPROVEMENTS

"Reshade" and its associated algorithm have established the validity of *in situ* computation of linear acoustic array optimal shading weights. Virtually no sea trial is conducted today without reshading to compensate for failed elements. Extensions to larger linear array problems are potentially useful. Improvements and modifications to the original source code are possible in the light of recent advances in signal processing hardware, and are needed to obtain reasonable computation times for these larger arrays. With the advent of single-board array processors, the beam pattern computations done (implicitly) in each iteration in the generic optimization model (KAPROX) may be performed more quickly and accurately using a floating point FFT. This is but one example of software modifications which will enhance the performance of "Reshade."

The generic nature of the optimization routine lends itself to the solution of more general array problems. These arrays may be multiline, planar, or three-dimensional with arbitrary geometry. Each geometry, however, will require a specific driver routine to set up the problem to be optimized. In general, the drivers would need the capability to address complex weights, allocate enough memory for computations, and to take into account any application-specific constraints imposed on the optimization problem. Additional constraints can be useful; for instance, constraints can sometimes be used in active arrays to control adverse effects of acoustic coupling between the array elements.

APPENDIX A

REPRINT PAPER: In Situ Optimal Reshading of Arrays with Failed Elements

In Situ Optimal Reshading of Arrays with Failed Elements

MICHAEL S. SHERRILL AND ROY L. STREIT, SENIOR MEMBER, IEEE

(Invited Paper)

Abstract—**An algorithm is presented which computes optimal weights for arbitrary linear arrays. The application of this algorithm to *in situ* optimal reshading of arrays with failed elements is discussed. It is shown that optimal reshading can often regain the original sidelobe level by slightly increasing the mainlobe beamwidth. Three examples are presented to illustrate the algorithm's effectiveness. Hardware and software issues are discussed. Execution time for a 25-element array is typically between 1 and 2 min on an HP9836C microcomputer.**

I. Introduction

A linear array of discrete elements (sensors) often experiences *element failures in situ*. These failures can significantly increase the sidelobe levels of the array wavenumber response, depending on how many elements fail and where the elements are located within the array. We discuss here an optimal reshading (reweighting) algorithm which can be applied *in situ* to reduce the sidelobe levels to the original design level. In many common element-failure situations, optimal reshading can regain the original sidelobe level by slightly increasing the mainlobe beamwidth. In arrays which experience significant element failures, optimal reshading is still possible, but may be of limited use. Three examples given below demonstrate a few of the possibilities.

An algorithm for optimal reshading was first proposed in [1] by Streit and Nuttall. Their algorithm utilized the general-purpose subroutine [2] to solve a specially structured "linear programming" problem. Unfortunately, their algorithm required hours of computation time and large amounts of computer storage on a minicomputer (the VAX 11/780) to optimally reshade a 50-element array with five failed elements. Consequently, their algorithm is not useful for *in situ* optimal reshading.

The shading algorithm proposed here differs from Streit and Nuttall's primarily in that we solve their linear programming problem using a new general-purpose subroutine [3], [4], herein referred to as Algorithm 635. Algorithm 635 uses the special structure of the linear programming problem to reduce time and storage requirements by orders of magnitude. Algorithm 635 can be incorporated easily in Streit and Nuttall's original approach. A significant algorithmic improvement was discovered in the course of this study and is described below. The resulting shading algorithm is fast enough and small enough to execute successfully on microcomputers (such as the HP9836C used here) in only a few minutes. Typical execution time for a 25-element array is under 2 min; for a 50-element array, execution time is typically under 10 min. The current algorithm, and the HP9836C with its inherent transportability, comprise an effective system for optimal reshading *in situ*.

Manuscript received March 11, 1986; revised August 11, 1986.

The authors are with the Naval Underwater Systems Center, New London, CT 06320.

IEEE Log Number 8714258

II. Optimal Array Shading

The wavenumber response of a linear array composed of N discrete omnidirectional elements located at arbitrary fixed positions x_n is given by

$$T(k) = \sum_{n=1}^{N} w_n \exp[-ikx_n] \tag{1}$$

where w_n are the element weights and the independent variable k denotes wavenumber in radians per unit length. The element weights are required to be real, but this entails no loss of generality (see below in Section III). Also, from (1), $T(-k) = T^*(k)$ for real weights (asterisk denotes conjugation), so it is unnecessary to consider negative values for k and we confine our attention to nonnegative k.

The array response as a function of k can be considered to be composed of a mainlobe beamwidth and a sidelobe region. The objective of the optimization process is to make $|T(k)|$ as small as possible on the user-specified sidelobe interval. Array weights which achieve this objective are said to be optimal. The optimization process usually produces equivalued sidelobes in the sidelobe region.

Weights that are optimal for a full array do not remain optimal after the array experiences element failures. To partially compensate for failed elements, the array is optimally reshaded by undertaking the optimization process again and incorporating knowledge of which elements have failed. As the examples below will show, the effectiveness of this strategy depends upon how many elements have failed and the location of these elements in the array.

The sidelobe interval is defined differently depending on the interelement spacing of the array. For an array with periodically spaced elements and no failures, the sidelobe interval is defined to be $[K_0, (2\pi/D) - K_0]$, where K_0 is calculated from the desired sidelobe level and the number N of array elements.[1] D is the physical distance from sensor to sensor.

[1] For an N-element array and $-t$-dB peak sidelobes, we have $K_0 = (2/D)\arccos(1/Z_0)$ where $2Z_0 = [r - (r^2 - 1)^{1/2}]^{1/M} + [r + (r^2 - 1)^{1/2}]^{1/M}$, $r = 10^{t/20}$, and $M = N - 1$. The interelement spacing D is assumed to be half of the so-called design wavenumber, and N is the number of array elements before failures.

Furthermore, the minimization interval can be reduced to $[K_0, \pi/D]$, since the response of this array is symmetric about $k = \pi/D$. K_0 is typically the point on the mainlobe response which is equal in magnitude to the peak of the sidelobes, but this is not always true for seriously degraded and/or aperiodic arrays (see Example 3 below). For arrays with aperiodically spaced elements, the sidelobe interval, denoted by $[K_0, K_1]$, must be chosen by inspection of a nonoptimal beam pattern or some other means. $|T(k)|$ must be minimized over the full $[K_0, K_1]$ range since, in general, an aperiodic array response is not symmetric about any wavenumber other than $k = 0$. The ability to specify arbitrary K_0 and K_1 is particularly useful for those applications involving aperiodically spaced elements because lower sidelobe levels may be obtained by looking at different minimization regions.

The optimization process deals with element failures in an array in the following way.

Step 1. Maintain mainlobe beamwidth and permit the sidelobe levels to rise.

Step 2. Regain, if possible, the original sidelobe level by broadening the mainlobe.

Broadening the mainlobe by increasing K_0 (step 2) is performed only if the sidelobe level, even after optimal reshading, has risen to an unacceptable value because of element failures. Thus step 1 is normal algorithmic procedure, and step 2 requires some iteration in specifying K_0 and/or K_1 because a compromise has to be made between the mainlobe beamwidth and the level of the sidelobes.

The solution of the array problem in the original formulation [1] is mathematically equivalent to solving an overdetermined system of complex linear equations. Unacceptably high sidelobes result if this system is solved in the usual least squares sense, so it is necessary to solve the system so that the magnitude of the maximum residual error is minimized. There now exists [3] an efficient algorithm and corresponding FORTRAN code [4] for solving problems of this sort to high accuracy.

To obtain the beamformer equation in an appropriate format to utilize this algorithm, we normalize the peak response of $T(k)$ so that $T(0) = 1$. This gives

$$\sum_{n=1}^{N} w_n = 1. \tag{2}$$

We solve (2) for the Nth weight w_N and substitute in (1) to obtain

$$T(k) = \exp[-ikx_N] + \sum_{n=1}^{N-1} w_n[\exp(-ikx_n) - \exp(-ikx_N)]. \tag{3}$$

By sampling $T(k)$ at the M equispaced points

$$k_m = K_0 + \frac{[K_1 - K_0]}{M-1}(m-1), \qquad m = 1, \cdots, M \tag{4}$$

we can write the problem of minimizing the peak sidelobe level of the array response as

$$\min_{w_n} \max_{1 \le m \le M} \left| f_m - \sum_{n=1}^{N-1} a_{mn} w_n \right| \tag{5}$$

where the complex numbers f_m and a_{mn} are defined by

$$f_m = \exp[-ik_m x_N]$$
$$a_{mn} = \exp[-ik_m x_n] - \exp[-ik_m x_N]. \tag{6}$$

The problem (5) is precisely the form necessary for application of Algorithm 635. For theoretical details of this algorithm, the interested reader is referred to [3].

Sometimes a few of the optimum weights for arrays with failed elements are observed to be negative, particularly those on the end elements. If the weights are applied in hardware, providing a 180° phase factor on the element output may not be desirable or possible. However, Algorithm 635 allows the selection of all nonnegative weights; this is implemented by the addition of constraints to (5). Usually, but not always, an element is zeroed if it would have had a negative weight. From (2) it follows that, if all the element weight values are required to be positive, they must be between 0 and 1. The requirement that weights $w_1, \cdots, w_N$ be between 0 and 1 can be written mathematically as

$$w_n - \frac{1}{2} \le \frac{1}{2}, \qquad n = 1, \cdots, N-1 \tag{7}$$

Algorithm 635 requires these $N - 1$ constraints. Algorithm 635 can also incorporate any number of general constraints of the form

$$\sum_{n} b_{mn} w_n - c_m \le d_m, \qquad m = 1, 2, \cdots, L \tag{8}$$

where c_m and d_m are constants. The requirement that w_N also be nonnegative gives

$$\left(1 - \sum_{n=1}^{N-1} w_n\right) - \frac{1}{2} \le \frac{1}{2}$$

or

$$\sum_{n=1}^{N-1} w_n - \frac{1}{2} \le \frac{1}{2} \tag{9}$$

which is clearly a special case of the general constraints (8).

III. Algorithm Improvements

Several changes to the algorithm presented in [1] enable significant reduction in the need for computational intensity. Lewis and Streit [5] have proved that, for a general line array shaded so that it has optimal sidelobe levels when steered through the same number of degrees either side of broadside, there exists a set of optimal weights that are real. Thus complex weights do not need to be considered. This fact allows an approximate eight-fold reduction in computation time and a two-fold reduction in storage requirements.

It is clear that the 50-element example run in Streit and Nuttall [1] was significantly oversampled in wavenumber. Their beam pattern can be reproduced with a four-fold reduction in the sampling of $T(k)$ (see Example 2 below), and this in no way detracts from the practical application of the algorithm. A significant reduction in computation time is realized by decreasing the number M of beam pattern samples in (4).

A significant algorithmic modification made to Algorithm 635 further decreases computation time. We have coined this modification "fast costing" and it is an important step in making the algorithm feasible on microcomputers such as the HP9836C. In order to describe this modification properly, some familiarity with the simplex method of linear programming and reference [3] is assumed.

Algorithm 635 can be broken into two fundamental computational operations called "costing" and "pivoting." "Costing" determines the so called minimum reduced cost coefficient and requires $2NM$ multiplications, where N is the number of discrete array elements and M is the number of samples taken of the beam pattern. "Pivoting" is a basis update and requires N^2 real multiplications. It is clear that the speed of the algorithm is intimately related to the number M of samples taken of the beam pattern, as well as the number N of discrete array elements. Since M is larger than N, "costing" requires more multiplications than "pivoting."

"Costing" in the linear array application means that, in each simplex iteration, the "discretized absolute value" of every sidelobe sample of the wavenumber response function $T(k_m)$, $m = 1, \cdots, M$, is computed to determine the "minimum reduced cost coefficient" of the current "basic feasible solution." By proceeding through a finite sequence of such "basic feasible solutions," we arrive at the solution of the "discretized problem." As shown in [3], this implies that the computed optimal wavenumber response function can have sidelobe levels that are theoretically at most 0.04 dB higher than the true optimum sidelobe level.[2] "Fast costing" refers simply to the fact that we first determine which of the sidelobe samples $T(k_m)$, $m = 1, \cdots, M$, has the largest true absolute value, and then compute the "discretized absolute value" of this one complex number. Therefore, only one "discretized absolute value" calculation is performed in each simplex iteration instead of M such calculations. The resulting reduction in computational effort is significant in microcomputing environments. The drawback is that the use of "fast costing" prevents the simplex algorithm from converging to a solution of the "discretized problem." Fortunately, however, it can be proved that we must approximate the solution in a well-defined sense. In the linear array application, "fast costing" results in the computed optimum beam pattern having sidelobe levels that are theoretically at most 0.08 dB higher than the true optimum level.[3] This is a small price to pay for major execution time improvements.

[2] The theoretical error of at most 0.04 dB is derived by taking $20 \log_{10} (\sec \pi/p)$, where $p = 32$. The term $\sec (\pi/p)$ is the error bound discussed in [3].

[3] Fast costing squares the error bound, giving $\sec^2 (\pi/p)$, or 0.08 dB when $p = 32$.

IV. Algorithm Implementation for *In Situ* Use

An algorithm must be reliable, easy to use, and fast when executing on portable microcomputers, to be useful for *in situ* application. The following section details the most important hardware and software issues addressed to enable *in situ* optimal reshading of arrays with failed elements.

The algorithm has been coded in BASIC and is comprised of Algorithm 635 and an array processing driver program. Algorithm 635 solves the linear program for a set of optimal weights, given data supplied by the driver program. The driver performs the initial setup based on several user inputs and provides all program output.

The driver program may be used with linear arrays having either periodic or aperiodically spaced elements. Program output consists of a graph of the optimal beam pattern, a graph of the optimal normalized element weights, and several parameters pertinent to the specific problem. Provision is made for storing the weights in a separate data file for possible use with digital beamformers.

A Hewlett-Packard (HP)-specific software modification was made by setting up the input data arrays (equation (6)) in buffers so that they are accessible for a one-dimensional multiply. For large-array dimensions, indexing a doubly subscripted data array and performing a dot product takes more time on the HP9836C than reading in a data array from a buffer, doing a MAT multiply, and performing a summation. (A MAT multiply is simply an element by element multiply of two equally dimensioned data arrays.) However, this procedure is more time consuming when the input data arrays are very small (i.e., the number of elements in the line array is small). The break even point occurs at around 12 or 13 elements, so it was decided to incorporate this speed enhancement for the longer running larger line arrays and trade off some speed reduction on the smaller line arrays.

To obtain fast execution times for *in situ* applications, we use one hardware speed enhancement, a 12.5-MHz fast CPU card with 16 kbytes of cache memory. This hardware supplement is available from HP for use on the HP9836C. Cache memory is fast memory resident on the CPU card for quick instruction acquisition. The use of the fast CPU board rather than the 8-MHz clock present in the standard computer configuration results in an approximate factor-of-two increase in observed speed.

The complete program is precompiled by use of software and a floating point math card available from the INFOTEK company. Precompilation reduces more computational portions of the BASIC code to machine language, giving an additional three-fold reduction in computation time. It is also desirable to upgrade the operating system for the HP to its latest revision. All work on these problems was run using the BASIC 3.0 operating system and the hardware supplements noted above.

Computation time is defined as time spent in Algorithm 635 and does not include the small amount of set-up time required by the driver program. Computation times are for the compiled BASIC program run on the HP9836C with the special hardware additions mentioned above.

The program described here needs just over 303 kbytes of

internal memory in addition to the memory required by the operating system to execute on the HP9836C. This is the amount of space required by fixing the maximum array size at $N = 50$, and allowing at most $M = 256$ beam pattern samples. Users can change dimensions to suit their specific needs, but storage requirements presently are directly proportional to the product NM. Even for a much larger number of line array elements, it is unlikely that memory restrictions would prove to be a problem on the HP9836C since extra memory boards of 1 Mbyte each are readily available.

Ongoing modifications should further enhance the capability and speed of the BASIC algorithm and driver. The addition of the ability to handle directional sensors is both useful and straightforward to implement. Execution of identical code on the new HP 300 series computers, which have a 16.6-MHz clock rate, should further reduce the computation time. Computation times on the order of 5 min for a 50-element array and 1 min for a 25-element array are anticipated.

It is possible to run the BASIC program in its uncompiled state. The execution of the program with cache memory and the fast CPU board as the only enhancements results in computation times of approximately 25 min for a 50-element array and 4.5 min for a 25-element array.

A copy of the entire program is available from the authors. Our specific implementation in HP BASIC utilizes several hardware and software devices to achieve computational efficiency, some of which may not be pertinent to other BASIC operating systems running on comparable machines. Users will undoubtedly find it necessary to make modifications to the code to allow it to run on other HP equipment or in BASIC on the VAX.

V. Examples

The following examples demonstrate the utility of the current algorithm for application *in situ* and provide insight into different situations that might arise when reshading equispaced arrays with failed elements. If optimal reshading can restore the array's original design sidelobe level by slightly increasing the mainlobe beamwidth, then we say that the optimal reshading has been effective. Optimal reshading is effective in many common element-failure situations. When the array is severely degraded, optimal reshading is less effective but is still useful in reducing the negative impact of element failures. These examples demonstrate that the effectiveness of reshading depends upon the number of element failures, as well as the location of the failed elements within the array.

Missing elements are modeled by zeroing the appropriate weights. In these examples, N refers to the number of intact array elements, M is the number of beam pattern samples, and K_0 is calculated by using the equation in an earlier footnote. We define the mainlobe width to be twice K_0 in all three examples.

A. Example 1: Effective Reshading

This example demonstrates that reshading can restore the original sidelobe level of an array response by slightly increasing the mainlobe beamwidth. In a 25-element equispaced array, originally designed for −30-dB sidelobes, elements 2 and 4 have failed. Therefore, $N = 23$, $M = 128$, and $K_0 = 0.6877$. We first keep the mainlobe width fixed and allow the sidelobe level to rise. See Fig. 1. The peak sidelobe level has risen to −26.86 dB below the mainlobe, and the mainlobe width is unchanged. If the sidelobe level after reshading is too high, an alternative to *discarding* or *repairing* the array is to broaden the mainlobe beamwidth. In Fig. 2, K_0 is increased to 0.775 and the peak sidelobe level diminishes to −30.04 dB below the mainlobe. A trade-off must always be made between an enlarged mainlobe beamwidth and an acceptable peak sidelobe level. In this case the mainlobe was increased 12.7 percent in order to recover the original sidelobe level. Execution times on the HP9836C are between 1 and 2 min for Figs. 1 and 2.

B. Example 2: Moderately Effective Reshading

This example is taken from Streit and Nuttall [1]. Because of the improvements detailed in Section III, above, the current algorithm runs faster on the HP9836C than on the VAX 11/780, although the floating point multiply time on the HP in its basic configuration is roughly 200 times slower than on the VAX.

Consider a linear array with 50 equispaced elements, initially designed for peak sidelobes of −30 dB relative to the mainlobe. Fig. 3 shows the classical Dolph–Chebyshev beam pattern with −30-dB sidelobes throughout the minimization range $[K_0, (2\pi/D) - K_0]$. This was computed using the current algorithm in 6.11 min. (This ideal case could have been computed analytically.)

Now we suppose that five elements, 7, 22, 40, 43, 50, of the array have failed. The optimal response after reshading the array is shown in Fig. 4. The peak sidelobe level has risen to −25.51 dB, but we have maintained mainlobe beamwidth and retained full steering capability. In this example $N = 45$ and $M = 128$.

This example (Fig. 4) took 7.47 minutes on the HP9836C and required 292 simplex iterations. The algorithm of Streit and Nuttall required 38.4 min and 402 iterations on the VAX.

Recovery of the original sidelobe level is possible (Fig. 5). The mainlobe beamwidth must be increased by the large factor 257.6 percent ($K_0 = 0.871$) and the execution of this task takes 8.98 min and requires 351 iterations. The constraint that all the weights lie between 0 and 1 is used. It is necessary to use the constraint in this instance because otherwise a dislocation of the maximum response from $k = 0$ results. This dislocation is due to the presence of too many negatively weighted elements.

C. Example 3: A Severely Degraded Array

This example shows that, for severely degraded arrays, recovery of the original sidelobe level may not be possible by increasing the mainlobe beamwidth, even after optimal reshading. Consequently, control of the level of the first sidelobe must be relinquished in order to gain control of the level of the *remaining* sidelobes.

Consider a 25-element array with elements 11 and 14 failed.

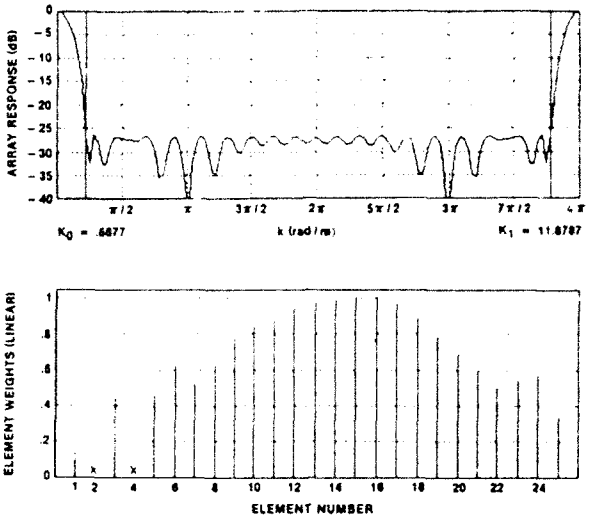


Fig. 1 Optimized array response and normalized weights for 25 elements with elements 2 and 4 missing

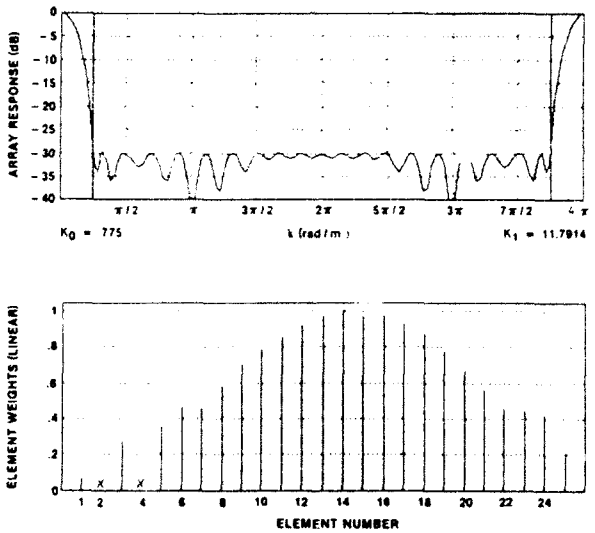


Fig. 2 Array response and normalized weights for Example 1 with K_0 = 0.775.

The original sidelobe level is −30 dB. Here $N = 23$, $M = 128$, and $K_0 = 0.6877$. Fig. 6 shows the algorithm's optimal response to this configuration [1] is a significant observation that, in this case, small perturbations of K_0 will not affect the level of the sidelobes. Only when the first sidelobe is incorporated into the mainlobe beamwidth ($K_0 = 1.27$) does the level of the remaining sidelobes return to the original desired value (see Fig. 7). It is apparent that decreasing the minimization interval by moving K_0 far enough to the right will improve the approximation, but one must give up control of the first sidelobe to reduce the others to acceptable levels. The net effect of losing two elements so close to the center is that negligible emphasis is placed on the remaining center elements (12 and 13) and the rest of the aperture is reshaded as if it were two separate arrays.

This situation cannot be overcome by using different

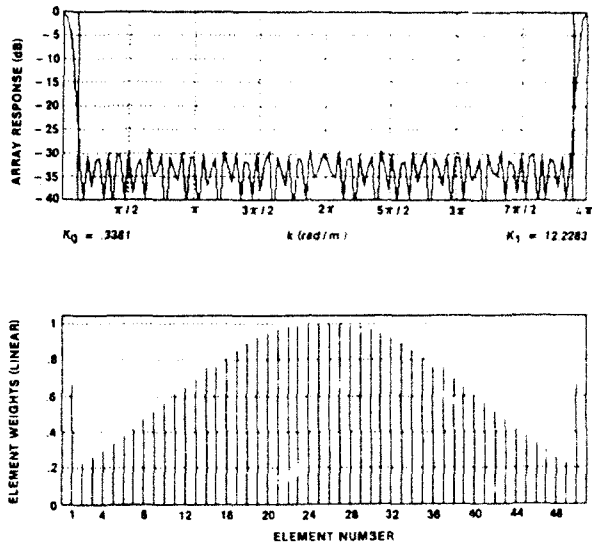


Fig. 3. Classical Dolph-Chebyshev array response and normalized weights for $N = 50$ and -30-dB sidelobes

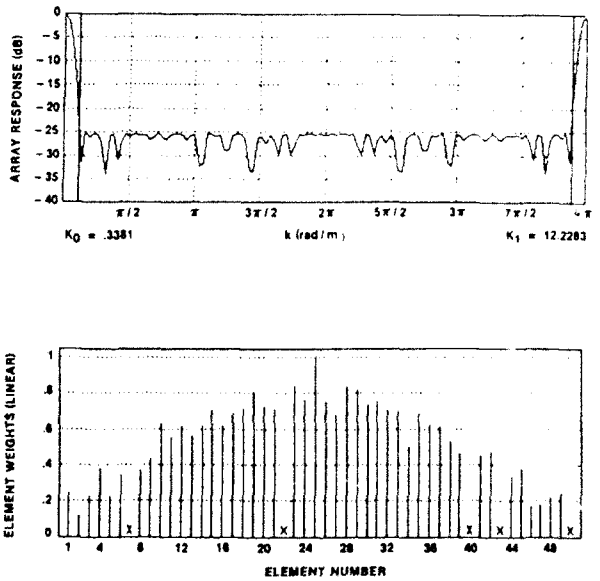
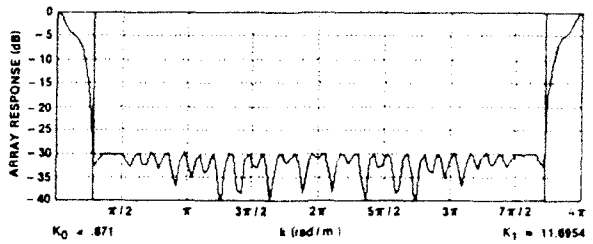


Fig. 4. Optimized array response and normalized weights for 50 elements with elements 7, 22, 40, 43, and 50 failed

weights. The optimal property of the array problem formulation and solution tells us that no weights exist which can suppress all the sidelobes below a certain level. Thus this array has lost too many elements and performance cannot be restored to its original design levels merely by reshading.

We have chosen to relinquish control of the first sidelobe to gain control of the level of the remaining sidelobes. We pick the first sidelobe merely for ease of implementation; modification of the algorithm to forfeit control of a different sidelobe could also have been done. The need to relinquish control of the first sidelobe level has only appeared in cases of severe array degradation due to element losses.



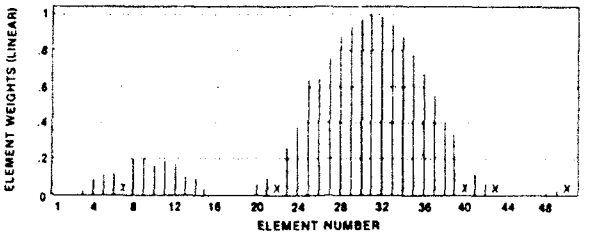
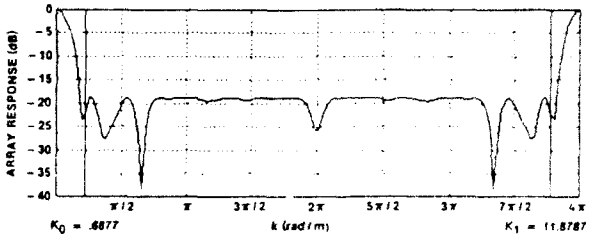


Fig. 5. Recovery of original sidelobe level. Example 2 with $K_0 = 0.871$.



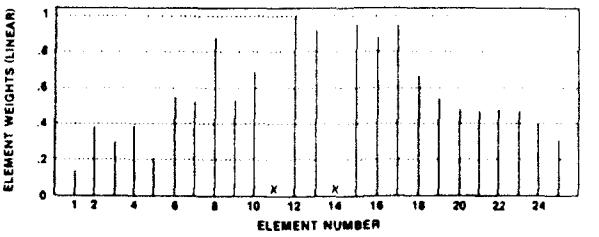


Fig. 6. Optimal array response and normalized weights for 25 elements with elements 11 and 14 failed.

VI. Conclusions

Arrays that have failed elements can be reshaded to obtain optimal array response functions. Optimal reshading is effective in many common element-failure situations. When the array is severely degraded, reshading is less effective, but still can be used to reduce the negative impact of element failures.

Optimal reshading can be accomplished *in situ*, quickly and reliably, on portable microcomputers using the algorithm described here. Arrays with 25 elements routinely run in less than 2 min and computation time for a 50-element array is less than 10 min. The algorithm can be applied to arrays of evenly or unevenly spaced linear geometry.

The above examples (and others) support the generally

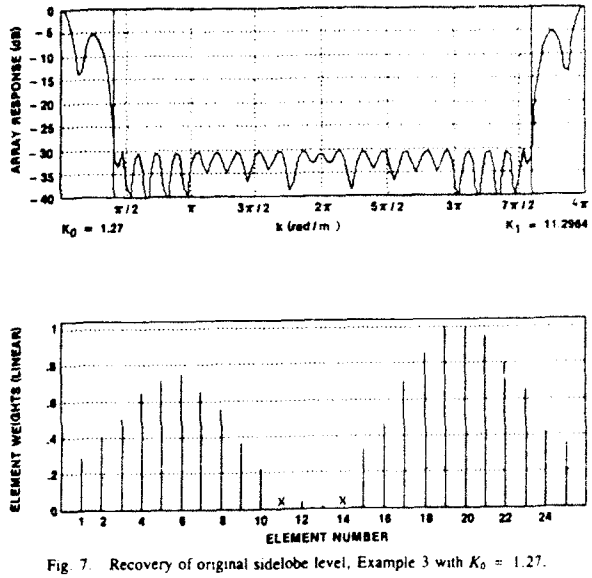


Fig. 7. Recovery of original sidelobe level, Example 3 with $K_0 = 1.27$.

accepted notion that failure of near-center elements is more detrimental to the array response than failure of near-edge elements.

Another application of Algorithm 635 is to arrays of planar and arbitrary three-dimensional geometry. Computation times for these more general arrays probably will depend upon N (number of sensors) and M (number of beam pattern samples) in the same manner as for linear arrays.

References

[1] R. L. Streit and A. H. Nuttall, "A general Chebyshev complex function approximation procedure and an application to beamforming," *J. Acoust. Soc. Amer.*, vol. 72, pp. 181–189, 1982.

[2] I. Barrodale and C. Philips, "Solution of overdetermined systems of linear equations in the Chebyshev norm," *Algorithm 495, ACM Trans. Math. Software*, vol. 1, pp. 264–270, 1975.

[3] R. L. Streit, "Solution of systems of complex linear equations in the L_∞ norm with constraints on the unknowns," *Soc. Indus. Appl. Math.*, vol. 7, no. 1, pp. 132–149, 1986.

[4] R. L. Streit, "An algorithm for the solution of systems of complex linear equations in the L_∞ norm with constraints on the unknowns," *Algorithm 635, ACM Trans. Math. Software*, vol. 11, no. 3, pp. 242–249, 1985.

[5] J. T. Lewis and R. L. Streit, "Real excitation coefficients suffice for sidelobe control in a linear array," *IEEE Trans. Antennas Propagat.*, vol. AP-30, pp. 1262–1263, 1982; also Naval Underwater Systems Center, New London, CT, Tech. Memo. 811114, Aug. 17, 1981.

Michael S. Sherrill was born on March 8, 1961, in Dover, DE. He received the B.S. degree in electrical engineering from the University of Delaware, Newark, in 1983.

He joined the staff of the Naval Underwater Systems Center, New London, CT, upon graduation. He is interested in mathematical modeling of physical systems and structured programming. Currently, he is responsible for a data acquisition and processing system for towed arrays.

★

Roy L. Streit (SM'84) was born in Guthrie, OK, on October 14, 1947. He received the B.A. degree (with honors) in mathematics and physics from East Texas State University, Commerce, in 1968, the M.A. degree in mathematics from the University of Missouri, Columbia, in 1970, and the Ph.D. degree in mathematics from the University of Rhode Island, Kingston, in 1978.

He is currently an Adjunct Associate Professor of the Department of Mathematics, University of Rhode Island. He was a Visiting Scholar in the Department of Operations Research, Stanford University, Stanford, CA, during 1981–1982. He joined the staff of the Naval Underwater Systems Center, New London, CT (then the Underwater Sound Laboratory), in 1970. He is an Applied Mathematician and has published work in several areas, including antenna design, complex function approximation theory and methods, and semi-infinite programming. He is currently conducting research in towed array design and hidden Markov models.

APPENDIX B

DRIVER PROGRAM LISTING: "Reshade"
Lines 1 through 481

```
1     OPTION BASE 1
2     OUTPUT 2 USING "#,B";255,75          ! CLEAR SCREEN
3     PRINTER IS CRT
4     RAD
5     REAL Estore(50),U(256),Tbeam(256),Grres(256),Hsens(256)
6     INTEGER Ioexit(10),Itlog(10),Icount(50),Ijswt(3,51)
7     INTEGER Ldim,I,J,K,N,CS,Svmflag,Cachflag,Floatflag,H
8     COM /Arrss/ Zradii(50),Bradii(4),Cheb(10),Z(50),Zcentr(50)
9     COM /Arrss1/ Ref(256),Imf(256),Reb(4,50) BUFFER,Imb(4,50) BUFFER,Rebcentr(
4),Imbcentr(4)
10    COM /Proj/ Basinv(51,54),Cossin(2,1025),Rea(256,50) BUFFER,Ima(256,50) BUF
FER,Cos45,Space
11    COM /Param/ INTEGER Ndim,M,L,Logp,Ndimp1,Ndimp4,Sil,Tme,Misel(10)
12    COM /Buffmult/ Colrea(50),Colima(50),Colreb(50),Colimb(50)
13    COM /Groups/ INTEGER Nogroup,REAL Senslen,Xgroup(25),Dgroup,D(50)
14    COM /Groups1/ Hydsen$[3],Hydro$[3]
15    DIM Sil$[3],Equi$[3],Weight$[3],Negwet$[3],Newko$[3],Data_msus$[10],Filena
me$[10],Wgtstore$[3],Group_space$[3]
16  !
17    Data_msus$=":INTERNAL"
18    Cachflag=0
19    ON ERROR GOTO 24          ! POSSIBLE ERRORS IF INTERFACE NOT PRESENT
20    CONTROL 32,1;1            ! IF CACHE MEMORY IS PRESENT IT WILL BE UTILIZED
21    OFF ERROR
22    STATUS 32,1;Stats
23    IF Stats THEN Cachflag=1
24    Floatflag=0
25    ON ERROR GOTO Redo
26    CONTROL 32,2;1       ! IF FLOATING POINT CARD PRESENT IT WILL BE UTILIZED
27    OFF ERROR
28    STATUS 32,2;Stats
29    IF Stats THEN Floatflag=1
30  !
31 Redo: !                              OBTAIN INPUT DATA
32    LOOP
33      IF Ndim=0 THEN
34        Ndim=16
35      ELSE
36        Ndim=Ndim+Tme
37      END IF
38      REPEAT
39        PRINT "ENTER TOTAL NUMBER OF ELEMENTS/GROUPS IN ARRAY: (3-50) ["&VAL$(
Ndim)&"]"
40        INPUT Ndim
41      UNTIL Ndim>2 AND Ndim<51
42  !
43      OUTPUT 2 USING "#,B";255,75        ! CLEAR SCREEN
44      REPEAT
45        PRINT "ENTER NUMBER OF SENSORS IN EACH GROUP: ["&VAL$(Nogroup)&"]"
46        INPUT Nogroup
47      UNTIL Nogroup<26
48      IF Nogroup=0 THEN Nogroup=1
49  !
50      IF Nogroup<>1 THEN
51        REDIM Xgroup(Nogroup)
52        OUTPUT 2 USING "#,B";255,75      ! CLEAR SCREEN
53        REPEAT
54          Group_space$=""
55          INPUT "IS ELEMENT SPACING WITHIN THE GROUP CONSTANT? (Y/N) [Y]",Grou
p_space$
56          IF LEN(Group_space$)=0 THEN
57            Group_space$="Y"
58          ELSE
59            Group_space$=UPC$(Group_space$[1])
60          END IF
61        UNTIL Group_space$="Y" OR Group_space$="N"
```

```
62   !
63        IF Group_space$="N" THEN
64          REPEAT
65            H=0
66            PRINT "ENTER POSITIONS OF SENSORS IN GROUP:"
67            FOR I=1 TO Nogroup
68              PRINT "SENSOR #"&VAL$(I)&":"
69              INPUT Xgroup(I)
70              IF I>1 AND Xgroup(I)<Xgroup(I-1) THEN H=H+1
71            NEXT I
72          UNTIL H=0
73        ELSE
74          REPEAT
75            PRINT "ENTER SPACING BETWEEN SENSORS IN GROUP: ["&VAL$(Dgroup)&"]"
76            INPUT Dgroup
77          UNTIL Dgroup>0
78          FOR I=1 TO Nogroup
79            Xgroup(I)=(I-1)*Dgroup
80          NEXT I
81        END IF
82      ELSE
83        MAT Grres= (1.)
84      END IF
85   !
86      OUTPUT 2 USING "#,B";255,75        ! CLEAR SCREEN
87      REPEAT
88        Hydsen$=""
89        PRINT "DO YOU WISH TO INCORPORATE A HYDROPHONE SENSITIVITY? (Y/N) [N]"
90        INPUT Hydsen$
91        IF LEN(Hydsen$)=0 THEN
92          Hydsen$="N"
93        ELSE
94          Hydsen$=UPC$(Hydsen$[1])
95        END IF
96      UNTIL Hydsen$="Y" OR Hydsen$="N"
97   !
98      IF Hydsen$="N" THEN
99        MAT Hsens= (1.)
100     ELSE
101       OUTPUT 2 USING "#,B";255,75      ! CLEAR SCREEN
102       REPEAT
103         PRINT "ENTER THE PHYSICAL SENSOR LENGTH: (METERS) ["&VAL$(Senslen)&"
]"
104         INPUT Senslen
105       UNTIL Senslen>0.
106  !
107       REPEAT
108         Hydro$=""
109         PRINT "IS HYDROPHONE TO BE MODELED AS A DIPOLE OR CONTINUOUS SENSOR?
 (D/C) [C]"
110         INPUT Hydro$
111         IF LEN(Hydro$)=0 THEN
112           Hydro$="C"
113         ELSE
114           Hydro$=UPC$(Hydro$[1])
115         END IF
116       UNTIL Hydro$="C" OR Hydro$="D"
117     END IF
118  !
119     OUTPUT 2 USING "#,B";255,75        ! CLEAR SCREEN
120     REPEAT
121       PRINT "ENTER TOTAL NUMBER OF MISSING ELEMENTS/GROUPS [ "&VAL$(Tme)&" ]
:"
122       INPUT Tme
123     UNTIL Tme>=0 AND Ndim-Tme>2
124  !
```

```
125     IF Tme>0 THEN
126       REDIM Misel(Tme)
127       REPEAT
128         OUTPUT 2 USING "#,B";255,75    ! CLEAR SCREEN
129         PRINT "ENTER MISSING ELEMENT/GROUP NUMBERS (SEPARATED BY COMMAS) [ "
;Misel(*);" ];"
130         INPUT Misel(*)
131         MAT SORT Misel(*)
132         H=0
133         FOR I=1 TO Tme
134           IF Misel(I)<1 OR Misel(I)>Ndim THEN H=H+1
135           IF I>1 THEN
136             IF Misel(I)=Misel(I-1) THEN H=H+1
137           END IF
138         NEXT I
139       UNTIL H=0
140     END IF
141   !
142     OUTPUT 2 USING "#,B";255,75               ! CLEAR SCREEN
143     REPEAT
144       INPUT "ARE ALL ELEMENTS/GROUPS EQUISPACED? (Y/N) [Y]",Equi$
145       IF LEN(Equi$)=0 THEN
146         Equi$="Y"
147       ELSE
148         Equi$=UPC$(Equi$[1])
149       END IF
150     UNTIL Equi$="Y" OR Equi$="N"
151   !
152     REDIM D(Ndim-Tme)
153 New_ko: !
154     Symflag=0                          ! FLAG FOR ARRAY SYMMETRY
155     IF Equi$="Y" THEN                  ! EQUISPACED ARRAY
156       OUTPUT 2 USING "#,B";255,75      ! CLEAR SCREEN
157       IF S11=0 THEN S11=30
158       REPEAT
159         PRINT "ENTER ORIGINAL SIDELOBE LEVEL (DB): (0 TO 50)) [-"&VAL$(S11)&
"]"
160         INPUT S11$
161         IF LEN(S11$)<>0 THEN S11=ABS(VAL(S11$))
162       UNTIL S11>-1 AND S11<51
163   !
164       OUTPUT 2 USING "#,B";255,75      ! CLEAR SCREEN
165       REPEAT
166         PRINT "ENTER ELEMENT/GROUP SPACING (METERS) (0-15) [ "&VAL$(Space)&"
 ]"
167         INPUT Space
168       UNTIL Space>0 AND Space<=15
169   !
170       N=Ndim-1
171       R=10^(S11/20)                ! CALCULATE K0
172       R2=R*R
173       R3=SQR(R2-1.)
174       R5=(R+R3)^(1./N)
175       R6=(R-R3)^(1./N)
176       Zo=(R5+R6)/2.
177       Ko=(2./Space)*ACS(1./Zo)
178       K1=2.*PI/Space-Ko
179   !
180       IF Newko$="Y" THEN
181         OUTPUT 2 USING "#,B";255,75    ! CLEAR SCREEN
182         REPEAT
183           PRINT "ENTER K0: [ "&VAL$(Ko)&" ]"
184           PRINT "SUGGESTED VALUE IS :";PROUND(Ko,-4)
185           INPUT Ko
186           K1=2.*PI/Space-Ko
187           IF Hydsen$="Y" OR Nogroup>1 THEN
```

```
188             PRINT "ENTER K1: [ "&VAL$(K1)&" ]"
189             INPUT K1
190           END IF
191         UNTIL Ko>0 AND Ko<PI/Space AND Ko<K1
192         Ndim=Ndim+Tme
193       END IF
194   !
195       C5=0
196       FOR I=1 TO Ndim
197         IF Tme>0 THEN
198           FOR J=1 TO Tme
199             IF I=Mise1(J) THEN 204
200           NEXT J
201         END IF
202         C5=C5+1
203         D(C5)=Space*(I-1)
204       NEXT I
205       CALL Symd(Ndim-Tme,Symflag,D(*))
206     ELSE
207       PRINT "ENTER ELEMENT/GROUP POSITIONS (METERS FROM END) :"
208       PRINT "SKIP MISSING ELEMENT/GROUP POSITIONS."
209       IF Newko$="Y" THEN Ndim=Ndim+Tme
210       FOR I=1 TO Ndim-Tme
211         REPEAT
212           H=0
213           PRINT "ELEMENT/GROUP "&VAL$(I)&" [ "&VAL$(D(I))&" ] :";
214           INPUT D(I)
215           IF I>1 THEN
216             IF D(I)<D(I-1) THEN H=H+1
217           END IF
218         UNTIL H=0
219         PRINT D(I)
220       NEXT I
221       OUTPUT 2 USING "#,B";255,75      ! CLEAR SCREEN
222       REPEAT
223         INPUT "ENTER KO: (RAD/METER)",Ko
224         INPUT "ENTER K1: (RAD/METER)",K1
225       UNTIL K1/Ko AND Ko>=0
226       CALL Symd(Ndim-Tme,Symflag,D(*))
227     END IF
228   !
229     Ndim=Ndim-Tme
230     Ndimp1=Ndim+1
231     Ndimp4=Ndim+4
232     IF Equi$="Y" THEN
233       M=64
234       C3=(K1-Ko)/(2.*M-1.)
235     ELSE
236       M=128
237       C3=(K1-Ko)/(M-1.)
238     END IF
239     Logp=5
240     P1=(2^Logp)+1
241   !
242     OUTPUT 2 USING "#,B";255,75        ! CLEAR SCREEN
243     REPEAT
244       Negwet$=""
245       INPUT "WILL YOU ALLOW NEGATIVE WEIGHTS ? (Y/N) [Y]",Negwet$
246       IF LEN(Negwet$)=0 THEN
247         Negwet$="Y"
248       ELSE
249         Negwet$=UPC$(Negwet$[1])
250       END IF
251     UNTIL Negwet$="Y" OR Negwet$="N"
252     IF Negwet$="Y" THEN          ! NO CONSTRAINTS: (Wj)<=1
253       L=0
```

```
254     ELSE                    ! 1 CONSTRAINT: (SUM(Wj)-.5)<=.5,(Wj-.5)>=.5
255       L=1                   ! CHANGE L AND REDIM APPROPRIATE ARRAYS
256     END IF                  ! FOR MORE CONSTRAINTS
257     Ldim=MAX(1,L)
258   !
259   ! REDIMENSION INPUT ARRAYS
260   !
261     REDIM Ioexit(Logp),Ijswt(3,Ndimp1),Itlog(Logp),Icount(Ndim),Zradii(Ndim)
262     REDIM Bradii(Ldim),Cheb(Logp),Estore(Ndim),Z(Ndim),Zcentr(Ndim)
263     REDIM Basinv(Ndimp1,Ndimp4),Cossin(2,P1),Rea(M,Ndim),Ima(M,Ndim)
264     REDIM Ref(M),Imf(M),Reb(Ldim,Ndim),Imb(Ldim,Ndim),Rebcentr(Ldim)
265     REDIM Imbcentr(Ldim),D(Ndim),U(M),Tbeam(2*M),Grres(M),Hsens(M)
266     REDIM Colrea(Ndim),Colima(Ndim),Colreb(Ndim),Colimb(Ndim)
267   !
268     MAT Basinv= (0.)          ! INITIALIZE COMMONS
269     MAT Cossin= (0.)
270     MAT Z= (0.)
271     MAT Cheb= (0.)
272     MAT Colrea= (0.)
273     MAT Colima= (0.)
274     MAT Colreb= (0.)
275     MAT Colimb= (0.)
276   !
277     IF Negwet$="Y" THEN
278       MAT Reb= (0.)
279       MAT Imb= (0.)
280       MAT Rebcentr= (0.)
281       MAT Imbcentr= (0.)
282       MAT Bradii= (0.)
283       MAT Zcentr= (0.)
284       MAT Zradii= (1.)
285     ELSE
286       MAT Reb= (1.)
287       MAT Imb= (0.)
288       MAT Rebcentr= (.5)
289       MAT Imbcentr= (0.)
290       MAT Bradii= (.5)
291       MAT Zcentr= (.5)
292       MAT Zradii= (.5)
293     END IF
294   !
295     FOR I=1 TO M
296       U(I)=Ko+C3*(I-1)              ! GENERATE U ARRAY
297     NEXT I
298   !
299     IF Hydsen$="Y" THEN             ! CALCULATE SENSITIVITY TERM
300       MAT Hsens= (0.)
301       IF Hydro$="D" THEN            ! DIPOLE SENSITIVITY
302         FOR J=1 TO M
303           Const=.5+.5*COS(U(J)*Senslen)
304           Hsens(J)=Const*Const
305           Const=-.5*SIN(U(J)*Senslen)
306           Hsens(J)=SQR(Hsens(J)+Const*Const)
307         NEXT J
308       ELSE                          ! CONTINUOUS SENSITIVITY
309         FOR J=1 TO M
310           Hsens(J)=ABS(SIN(U(J)*Senslen/2.)/(U(J)*Senslen/2.))
311         NEXT J
312       END IF
313     END IF
314   !
315     IF Nogroup<>1 THEN              ! CALCULATE GROUP RESPONSE
316       MAT Grres= (0.)
317       FOR J=1 TO M
318         Grim=0.
319         FOR I=1 TO Nogroup
```

```
320           Grres(J)=Grres(J)+COS(U(J)*Xgroup(I))
321           Grim=Grim-SIN(U(J)*Xgroup(I))
322         NEXT I
323         Grres(J)=SQR(Grres(J)*Grres(J)+Grim*Grim)/Nogroup
324       NEXT J
325     END IF
326   !
327     FOR J=1 TO M
328       Ref(J)=COS(D(Ndim)*U(J))           ! GENERATE F ARRAY
329       Imf(J)=-SIN(D(Ndim)*U(J))
330       FOR I=1 TO Ndim
331         Rea(J,I)=Ref(J)-COS(D(I)*U(J))   ! GENERATE Hk ARRAY
332         Ima(J,I)=Imf(J)+SIN(D(I)*U(J))
333         Rea(J,I)=Rea(J,I)*Grres(J)*Hsens(J)
334         Ima(J,I)=Ima(J,I)*Grres(J)*Hsens(J)
335       NEXT I
336     NEXT J
337   !
338     FOR I=1 TO M
339       Ref(I)=Ref(I)*Grres(I)*Hsens(I)
340       Imf(I)=Imf(I)*Grres(I)*Hsens(I)
341     NEXT I
342   !
343     N=Ndim-1
344     Itlog(1)=20*N                  !MAX ITERATION COUNT
345     Ioexit(1)=0                    !PRINT OPTION
346     Ts=TIMEDATE                    !INITIALIZE TIME
347     CALL Kaprox(N,Itlog(*),Ioexit(*),Ijsut(*))
348     Te=TIMEDATE-Ts                 !EXECUTION TIME
349     Estore(N)=Cheb(Logp)
350     Icount(N)=Itlog(Logp)
351   !
352     Zsum=0.                  ! CALCULATE FINAL WEIGHT=1-SUM OF ALL OTHERS
353     Zsum=SUM(Z)
354     Z(Ndim)=1.-Zsum
355   !
356     IF Symflag THEN                    ! SYMMETRIZE WEIGHTS
357       FOR I=1 TO INT((Ndim)/2)
358         P=(Z(I)+Z(Ndim-I+1))/2
359         Z(I)=P
360         Z(Ndim-I+1)=P
361       NEXT I
362     END IF
363   !
364     IF Tme>0 THEN
365       REDIM Z(Ndim+Tme)
366       FOR I=1 TO Tme
367         FOR J=Ndim+I TO Misel(I) STEP -1
368           IF J>Misel(I) THEN Z(J)=Z(J-1)
369         NEXT J
370         Z(Misel(I))=0.
371       NEXT I
372     END IF
373   !
374     IF Equi$="Y" THEN
375       CALL Calcbeam(Ndim,M,Tme,Misel(*),Space,Z(*),Tbeam(*))
376     ELSE
377       CALL Unsymbeam(Ndim,M,Tme,Misel(*),Ko,K1,Z(*),Tbeam(*))
378     END IF
379   !
380     Zmax=MAX(Z(*))
381     IF Zmax<>0 THEN
382       FOR I=1 TO Ndim+Tme              ! NORMALIZE WEIGHTS TO 1
383         Z(I)=Z(I)/Zmax
384       NEXT I
385     END IF
```

```
386   !
387       CALL Weight_plot(Ndim+Tme,M/2,2,Tme,Missl(*),Space,Ko,K1,C3,Z(*),Tbeam(*
),Equi$)
388       OUTPUT 2 USING "#,B";255,75
389       PRINTER IS PRT
390       PRINT USING "@"
391       DUMP GRAPHICS
392   !
393       CONTROL CRT,12;2
394       FOR I=0 TO 9
395         ON KEY I LABEL "" GOSUB Dummy
396       NEXT I
397       ON KEY 1 LABEL " CONTINUE " GOTO Comp
398       ON KEY 2 LABEL " KEYS OFF/ON " GOSUB Flip_key
399       LOOP
400       END LOOP
401 Dummy: !
402       RETURN
403 Flip_key: !
404       Keflip=(Keflip+1) MOD 2
405       IF Keflip THEN
406         CONTROL CRT,12;1
407       ELSE
408         CONTROL CRT,12;2
409       END IF
410       RETURN
411 Comp: !
412       GRAPHICS OFF
413       OFF KEY
414       Weight$=""
415       REPEAT
416         INPUT "WOULD YOU LIKE A LIST OF THE WEIGHTS? (Y/N) [Y]",Weight$
417         IF LEN(Weight$)=0 THEN
418           Weight$="Y"
419         ELSE
420           Weight$=UPC$(Weight$[1])
421         END IF
422       UNTIL Weight$="Y" OR Weight$="N"
423   !
424       CALL Printinputs(Cachflag,Floatflag,Ndim,Tme,Sll,Itlog(*),Logp,Missl(*),
Space,Ko,K1,Te,Cheb(*),Z(*),Equi$,Weight$,Negwet$)
425   !
426       OUTPUT 2 USING "#,B";255,75        ! CLEAR SCREEN
427       Newko$=""
428       REPEAT
429         INPUT "WOULD YOU LIKE TO CALCULATE A NEW ko or K1 TO GIVE A DIFF. BEAM
WIDTH (Y/N) [N]",Newko$
430         IF LEN(Newko$)=0 THEN
431           Newko$="N"
432         ELSE
433           Newko$=UPC$(Newko$[1])
434         END IF
435       UNTIL Newko$="Y" OR Newko$="N"
436       IF Newko$="Y" THEN GOTO New_ko
437   !
438       Wgtstore$=""
439       REPEAT
440         INPUT "WOULD YOU LIKE TO STORE THE WEIGHTS IN A DATA FILE? (Y/N) [N]",
Wgtstore$
441         IF LEN(Wgtstore$)=0 THEN
442           Wgtstore$="N"
443         ELSE
444           Wgtstore$=UPC$(Wgtstore$[1])
445         END IF
446       UNTIL Wgtstore$="Y" OR Wgtstore$="N"
447       IF Wgtstore$="Y" THEN
```

```
448       OUTPUT 2 USING "#,B";255,75      ! CLEAR SCREEN
449       INPUT "ENTER FILENAME FOR WEIGHT FILE: (10 CHARACTERS  ",Filename$
450       OUTPUT 2 USING "#,B";255,75      ! CLEAR SCREEN
451       INPUT "ENTER MASS STORAGE DEVICE: [ :INTERNAL ]",Data_msus$
452       IF Equi$="Y" THEN
453         IF Ndim>30 THEN
454           CREATE BDAT Filename$&Data_msus$,2,256
455         ELSE
456           CREATE BDAT Filename$&Data_msus$,1,256
457         END IF
458         ASSIGN @Stordat TO Filename$&Data_msus$
459         OUTPUT @Stordat;Ndim+Tme,Space,Z(*)
460       ELSE
461         SELECT Ndim
462         CASE >47
463           CREATE BDAT Filename$&Data_msus$,4,256
464         CASE >31
465           CREATE BDAT Filename$&Data_msus$,3,256
466         CASE >15
467           CREATE BDAT Filename$&Data_msus$,2,256
468         CASE ELSE
469           CREATE BDAT Filename$&Data_msus$,1,256
470         END SELECT
471         ASSIGN @Stordat TO Filename$&Data_msus$
472         OUTPUT @Stordat;Ndim+Tme,D(*),Z(*)
473       END IF
474       ASSIGN @Stordat TO *
475   !
476     END IF
477     GRAPHICS ON
478     PAUSE
479     GRAPHICS OFF
480   END LOOP                    ! RETURNS TO Redo AT PROGRAM BEGINNING
481   END
```

APPENDIX C

FLOPPY DISK: Program "Reshade"